Wright State University

Report-P7

Paul Fuchs

U00747698

CEG 3900 Mobile and Cloud Computing

Prabhaker Mateti

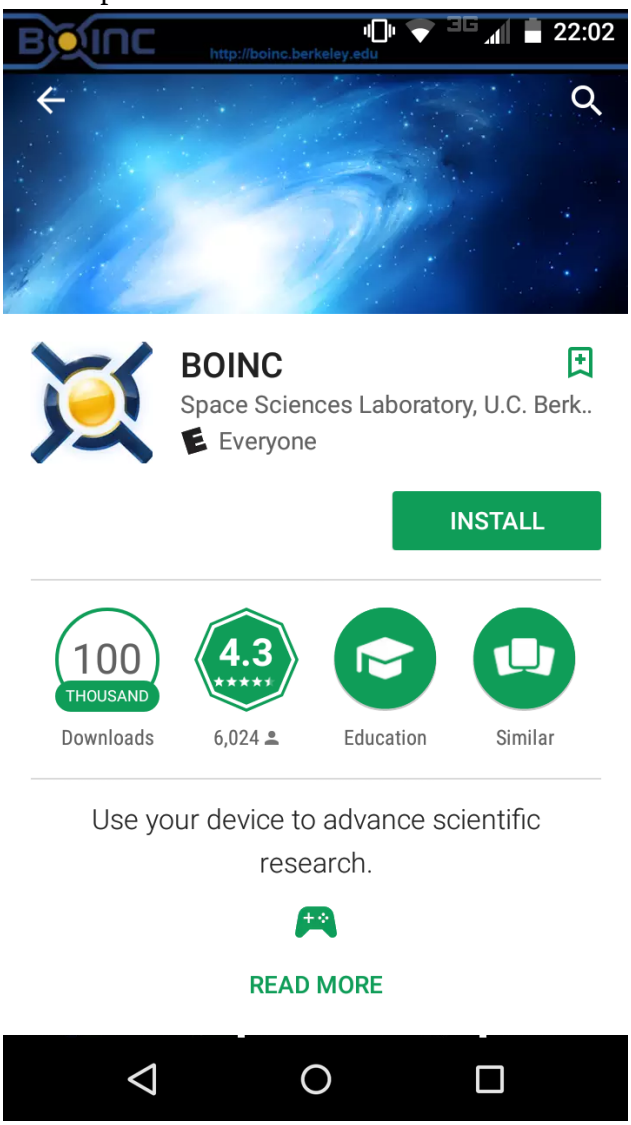
24 April 2017

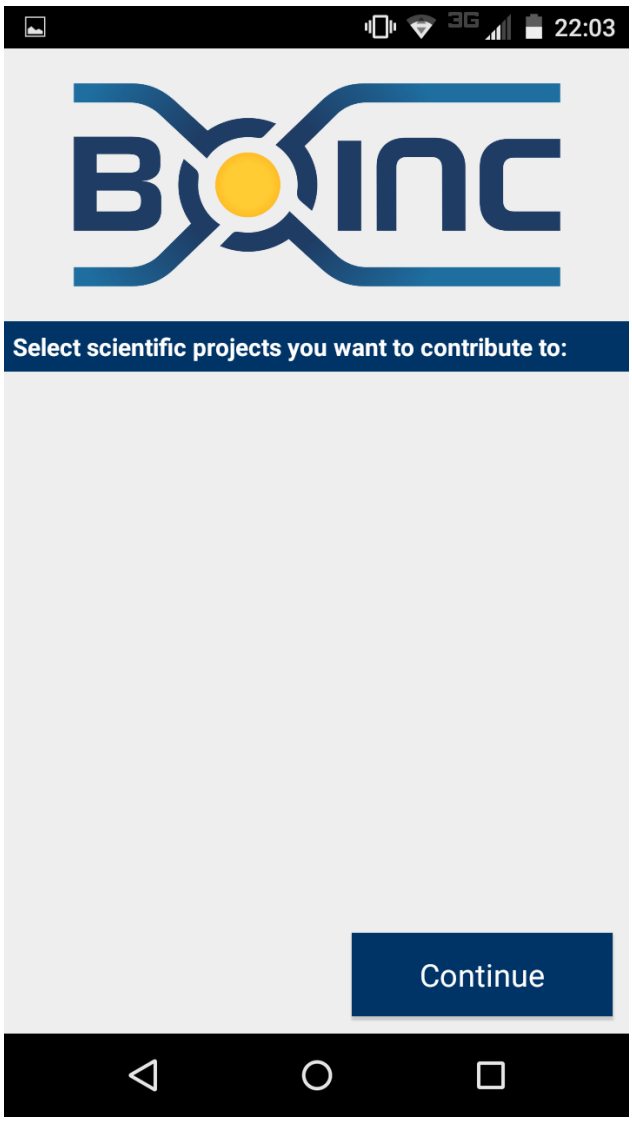
https://github.com/helmhammerhand/ceg-3900-spring-2017/

Table of Contents

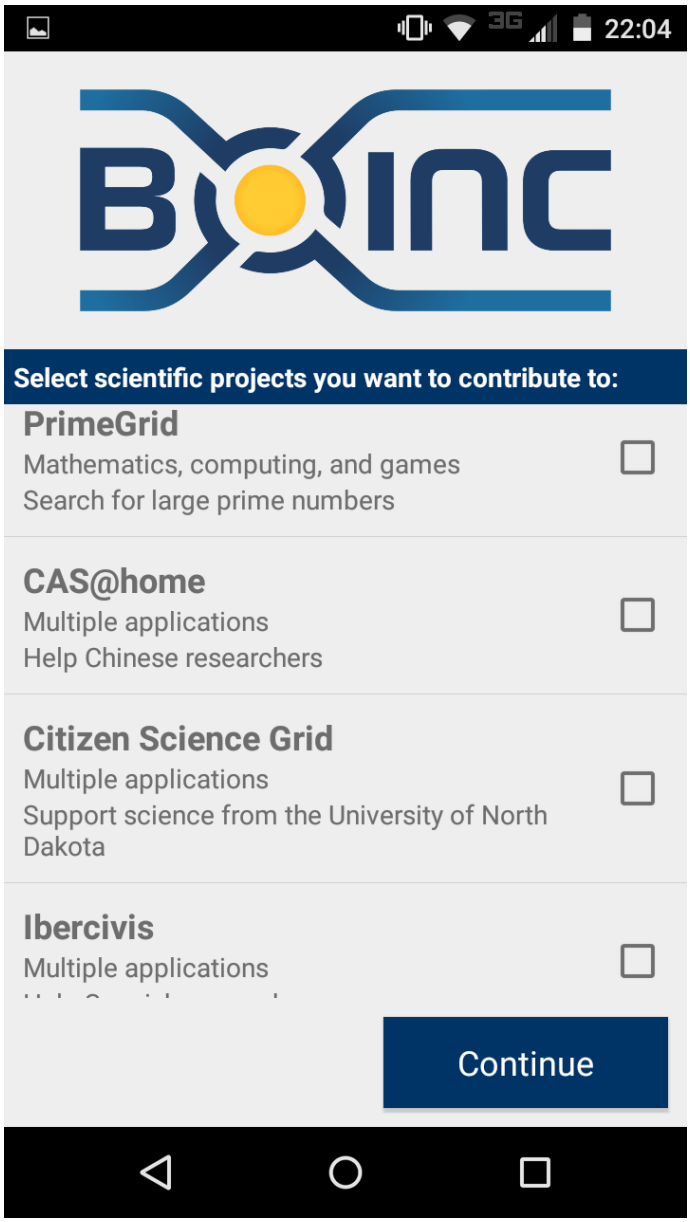
3.1 SETI@home (½ hour)

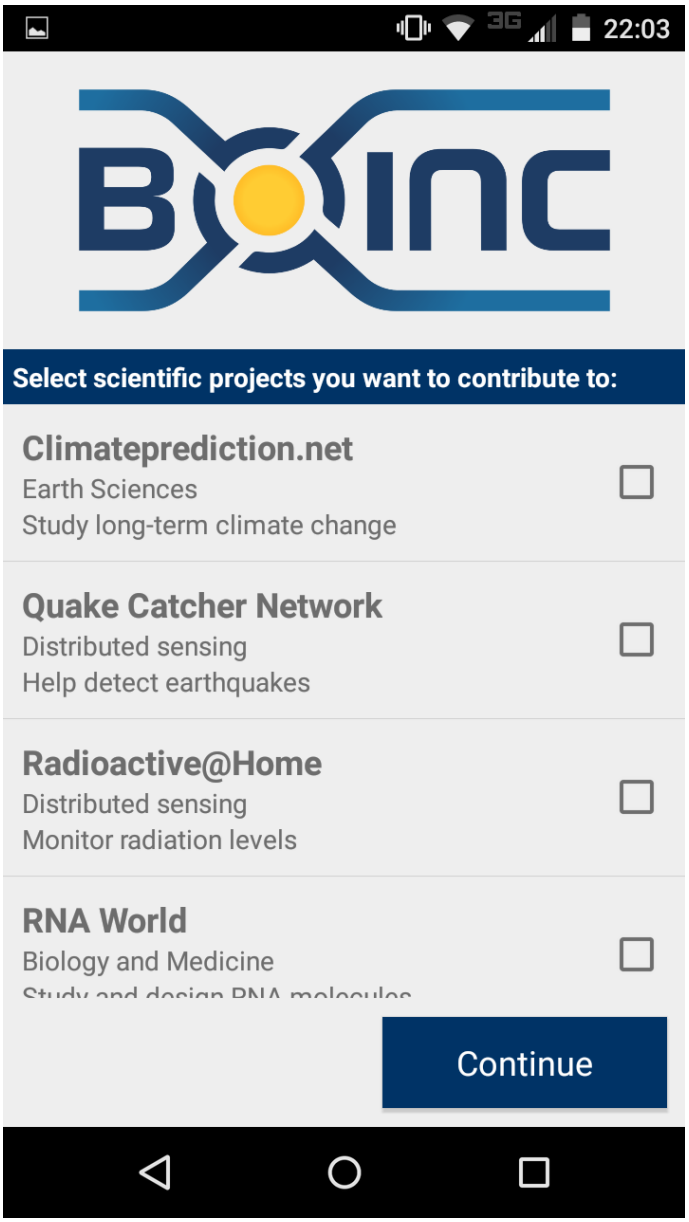
I installed the BOINC APK through google play. Running it brought up a screen with a continue button which I pressed.



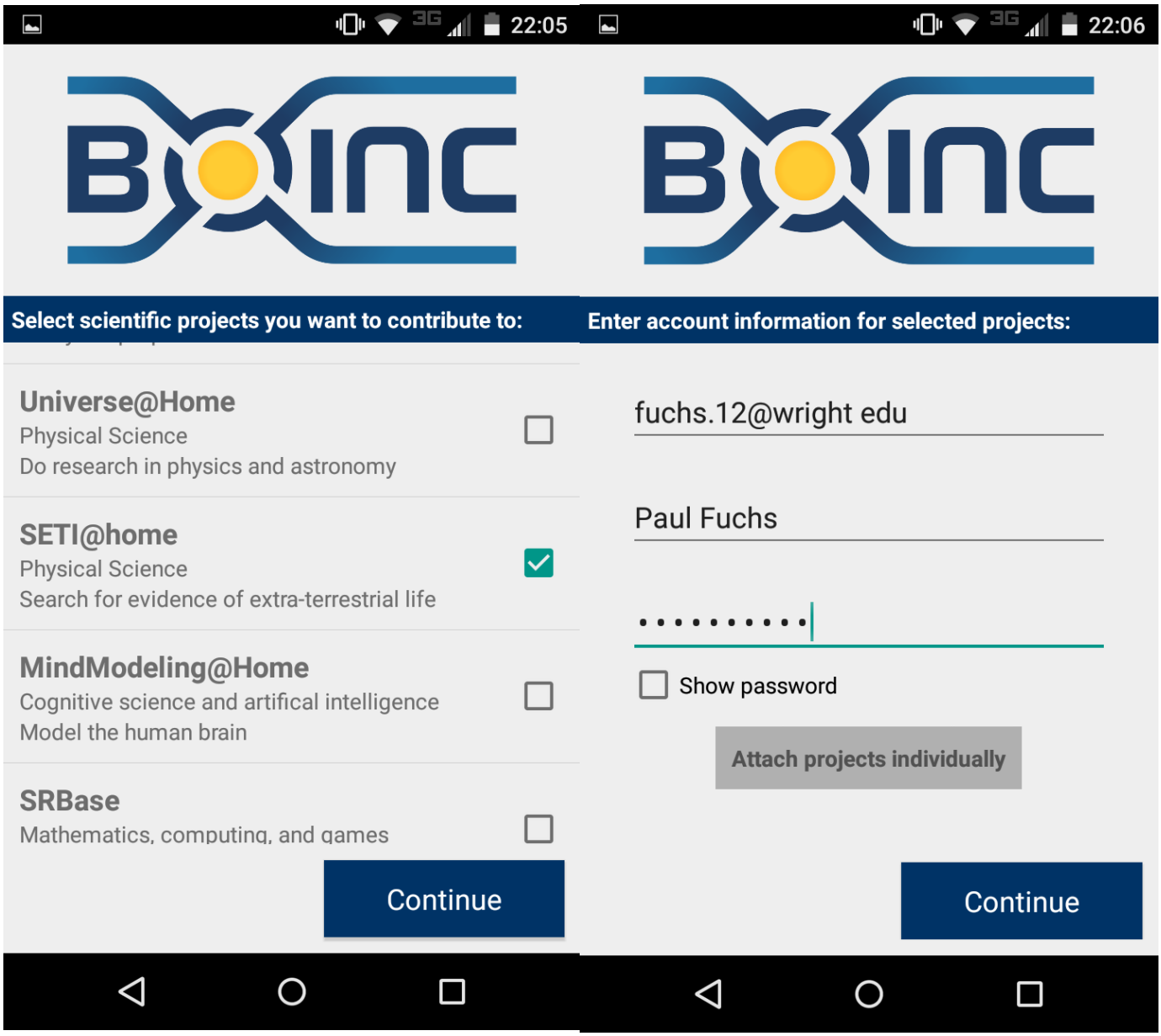


This brought up a screen with a surprising number of scientific projects from medical research to physics, to mathematics.

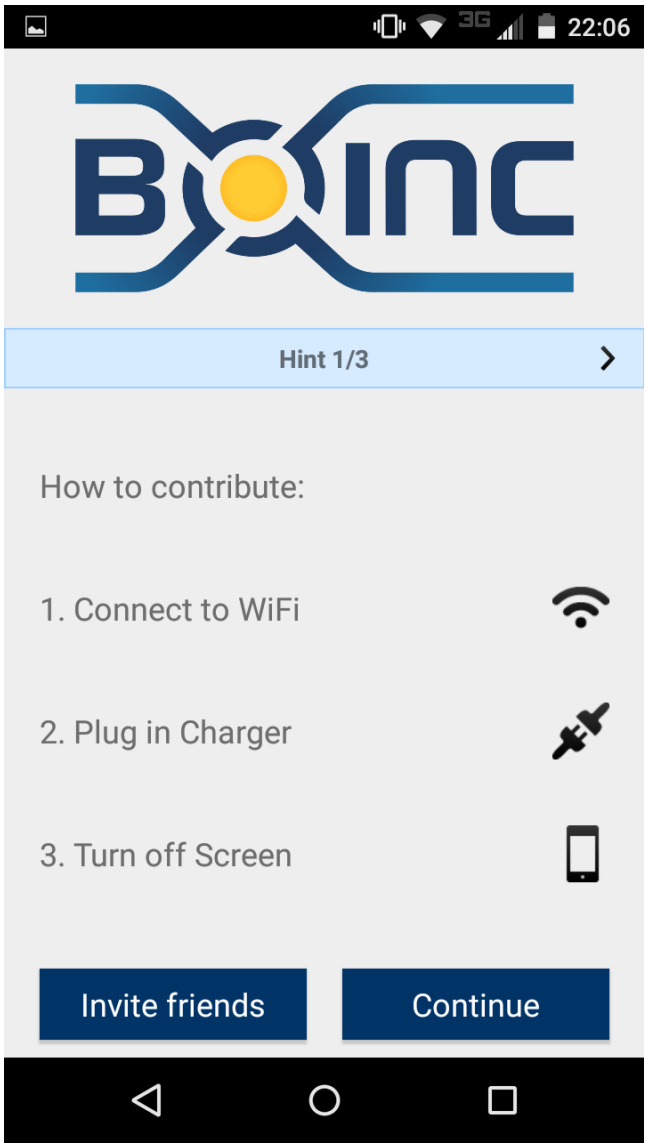


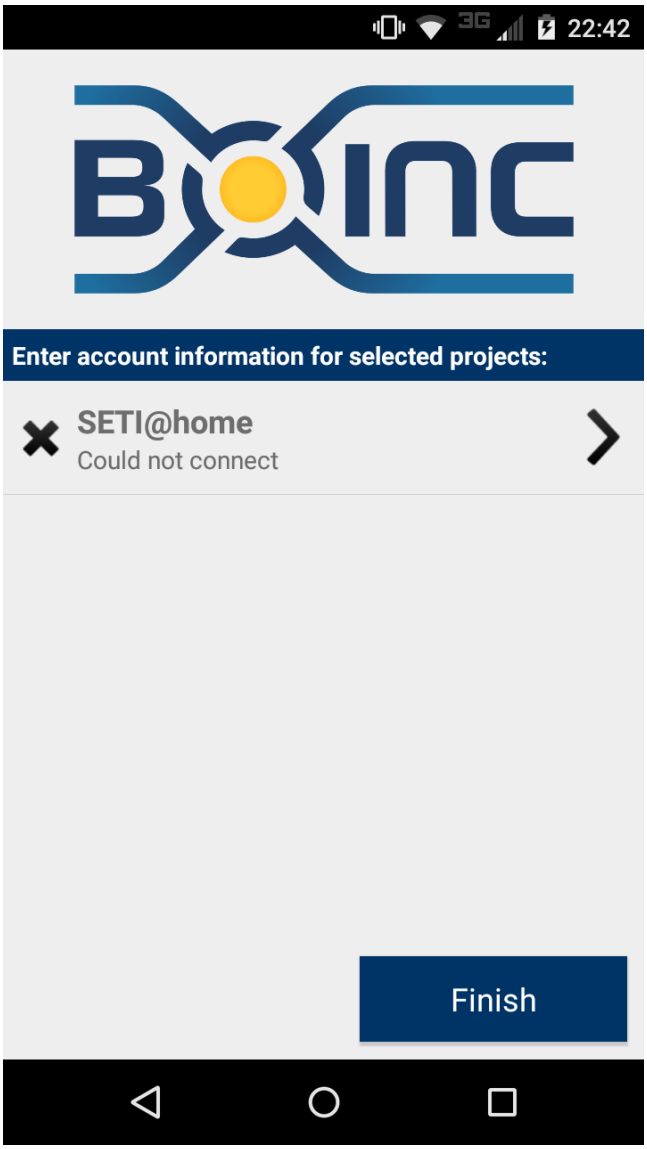


I selected the SETI@home project. It had me enter my email and name and a password as shown on the left.



The next screen displayed use tips. The next screen I can't quite figure out. It displays that SETI@home could not connect and I'm not sure why. However, when I unlock my screen, there is a notifaction for BOINC so it may actually be running. I can't get a picture of this notification because it disappears as soon as I unlock my phone.





3.2 Worldwide grade inflation (4 hours)

Java Server on AWS

Grade Averager Design:

File: GradeAverage
Input: directory containing the course files
Output: a string containing each country code and its average grade, one per line, sorted in ascending country order.

Algorithm Pseudocode:

Method: computeAverageByCountry(File)
List all files in the input directory
Filter files to removy any directories from the list
Create hash table that will store the current average and number of grades for each country
    The number of grades allows new datapoints to be added to the average without storing every datapoint according to the formula newAverage = (oldCount * oldAverage + newData)/(oldCount + 1). This functionality is implemented using the class AverageData

For each file:
    Get the country code by isolating the first four characters in the name
    Compute the average grade of the file:
        Method: computeCourseAverage(File)
        Initialize an AverageData to store the running average
        For each line in the file
            Get the student's grade by splitting the string on whitespace
            add the grade to the running average
        return the running average
    Add average of the file to the hash table
        If hash table already has an entry for the country
            merge the two averages using avg = (avg1 * cnt1 + avg2 * cnt2)/(cnt1+cnt2)
        Otherwise, put the file's average directly into the table
Get key set of the hash table and sort alphabetically
For each key in this sorted list:
    Print the country code followed by the average

Server Backend

I was not sure what the P7 spec meant by the APK passing a github link to the server since the provided zip file was on google drive. Because of this, I had the server receive a google drive url from the android client. The server uses wget to download this file from google drive and tar to unzip it. It then calls the computeAverageByCountry method explained above.

Using wget to retrieve the file from googledriveurl

```
//start the process
String cmd = "wget "+googledriveurl+ " -O gradezip.tar.bzip2";
cmd = cmd.replaceAll("&", "\\&");
System.out.println(cmd);
Process download = Runtime.getRuntime().exec(cmd);

//read the output from process to block it
Scanner s = new Scanner(download.getErrorStream());
String output = "";
while(s.hasNextLine())
        output += s.nextLine() + "\n";
System.out.println(output);
System.out.println("Download Complete");
```

Using tar to unzip the grades file

```
System.out.println("Unzipping");
//unzip the file
Process unzip = Runtime.getRuntime().exec("tar -vxjf gradezip.tar.bzip2");
//read the output from process to block it
s = new Scanner(unzip.getErrorStream());
output = "";
while(s.hasNextLine())
        output += s.nextLine() + "\n";
System.out.println(output);
System.out.println("Extraction complete");
```

The code for analyzing the grades directory is explained above. The implementation can be found on the github page.

Android APK Frontend

The APK is the same format as the CloudCat APK from 3.3 of P6. It has a screen with some text inputs and a button that triggers a service which queries the server and displays output in a scrooling text view at the bottom of the screen.

Since the grade file is not on github, the android APK takes the google drive id of the grade zip file from the user. This ID is converted to the download link using the url: **https://docs.google.com/uc?export=download&id=ID**

```
serviceIntent.putExtra("driveurl", "https://docs.google.com/uc?
export=download&id="+googleID.getText().toString());
```

I later discovered I had to escape the amperstand before calling wget in order for it tow work properly. I did this using url.replace(“&”, “\\&”);

This url is sent to the server which downloads, unzips, and analyzes the file. This takes quite some time (several minutes). Also, I sometimes get an error when google drive gives me a html file instead of the bzip2 file. This has something to do with google drive giving me a virus scanner warning. I’m not sure what triggers this, but I suspect it gives the warning when I try to download the file too many times in rapid succession.

Screenshots

Running APK. The first text field is the google drive ID and the second is the server's hostname. I ran this on my laptop instead of AWS to save time. It would work the same on AWS except the hostname would be different.

The url given in the lab is https://drive.google.com/file/d/0BxhMyqQW9vzoaHBOdzFnc1IyM3M/view. From this, I determined the file ID to be 0BxhMyqQW9vzoaHBOdzFnc1IyM3M

Server output

Server receives url and invokes the wget command below to download the file

```
Accpted Connection

Received Request:
https://docs.google.com/uc?export=download&amp;confirm=QBiP&amp;&id=0BxhMyqQW9vzoaHBOdzFnc1IyM3M
Downloading
wget https://docs.google.com/uc?export=download&amp;confirm=QBiP&amp;&id=0BxhMyqQW9vzoaHBOdzFnc1IyM3M
 -O gradezip.tar.bzip2
```

Output of wget

```
Downloading
wget https://docs.google.com/uc?export=download&amp;confirm=QBiP&amp;&id=0BxhMyqQW9vzoaHBOdzFnc1IyM3M
 -O gradezip.tar.bzip2
--2017-04-24 01:16:42--  https://docs.google.com/uc?export=download&amp;confirm=QBiP&amp;&id=0BxhMyqQ
W9vzoaHBOdzFnc1IyM3M
Resolving docs.google.com (docs.google.com)... 192.232.16.117, 192.232.16.91, 192.232.16.110, ...
Connecting to docs.google.com (docs.google.com)|192.232.16.117|:443... connected.
HTTP request sent, awaiting response... 302 Moved Temporarily
Location: https://doc-0o-60-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7deffksulhg5h7mbp1
/22iocvod3coj0d9mhi8k88mcmkdkujdi/1493006400000/06800180781796148841/*/0BxhMyqQW9vzoaHBOdzFnc1IyM3M?e
=download [following]
Warning: wildcards not supported in HTTP.
--2017-04-24 01:16:43--  https://doc-0o-60-docs.googleusercontent.com/docs/securesc/ha0ro937gcuc7l7de
ffksulhg5h7mbp1/22iocvod3coj0d9mhi8k88mcmkdkujdi/1493006400000/06800180781796148841/*/0BxhMyqQW9vzoaH
BOdzFnc1IyM3M?e=download
Resolving doc-0o-60-docs.googleusercontent.com (doc-0o-60-docs.googleusercontent.com)... 216.58.216.9
7, 2607:f8b0:4009:813::2001
Connecting to doc-0o-60-docs.googleusercontent.com (doc-0o-60-docs.googleusercontent.com)|216.58.216.
97|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/x-bzip2]
Saving to: ‘gradezip.tar.bzip2’
```

Server unzips the file using tar

```
Download Complete
Unzipping
```

Server analyzes each file in the WorldWideGrades directory and outputs averages

```
Averaging: ./WorldWideGrades/0044-2013-2709-IFYP6183
Averaging: ./WorldWideGrades/0092-2008-WNTR-CSCC1023
Averaging: ./WorldWideGrades/0260-2011-7665-FACL8189
Averaging: ./WorldWideGrades/0250-2013-SUMR-CSCC1012
Averaging: ./WorldWideGrades/0256-2012-8087-UGLY3243
Averaging: ./WorldWideGrades/0053-2006-FALL-CORS0032
Averaging: ./WorldWideGrades/0092-2013-SUMR-CSCC1003
Averaging: ./WorldWideGrades/0371-2007-WNTR-CORS7028
Averaging: ./WorldWideGrades/0218-2002-SPRN-CORS4121
Averaging: ./WorldWideGrades/0370-2015-WNTR-CORS4376
Averaging: ./WorldWideGrades/0040-2015-SPRG-CSCC1015
Found 80 countries
80

Sending Response:
0001    2.5079353294889812
0020    2.4829128626661046
0030    2.501585150665633
0032    2.5089838554561985
0033    2.4870004742065457
0034    2.036372673155058
0036    2.4974230585462203
0039    2.497121913287566
0040    2.035250417665431
0043    2.4924385254121693
0044    2.4939384683133037
0045    2.4913333895873704
0046    2.041865093711351
0047    2.0307461057819225
0048    2.031700634994871
0049    2.501967421837709
0051    2.034138670095904
0052    2.4763986205815525
0053    2.481633565288579
0055    2.4982939991787725
0056    2.509487429972687
0060    2.4870613782236104
0061    2.494989167502511
0066    2.040048856862596
0081    2.496606897331097
0084    2.498884713387893
0086    2.4839092137445045
0090    2.49479353072904
0091    2.5007677698723674
0092    2.0356616677508352
0098    2.5025838929126967
0212    2.4911110473347824
0213    2.501618815026604
0216    2.0382796600257325
0218    2.4789495458562922
0224    2.5064794714133463
0231    2.4989827675638243
0233    2.498871601231687
0234    2.0361697974679025
0244    2.510779611078376
0249    2.043944390187857
0250    2.0360839397830977
0251    2.475253803572047
0252    2.0473689884871553
0254    2.4852443077628297
```

Complete List of Averages

The average for each country is given below in the format “countryID average”

0001 2.5079353294889812

0020 2.4829128626661046

0030 2.501585150665633

0032 2.5089838554561985

0033 2.4870004742065457

0034 2.036372673155058

0036 2.4974230585462203

0039 2.497121913287566

0040 2.035250417665431

0043 2.4924385254121693

0044 2.4939384683133037

0045 2.4913333895873704

0046 2.041865093711351

0047 2.0307461057819225

0048 2.031700634994871

0049 2.501967421837709

0051 2.034138670095904

0052 2.4763986205815525

0053 2.481633565288579

0055 2.4982939991787725

0056 2.509487429972687

0060 2.4870613782236104

0061 2.494989167502511

0066 2.040048856862596

0081 2.496606897331097

0084 2.498884713387893

0086	2.4839092137445045
0090	2.49479353072904
0091	2.5007677698723674
0092	2.0356616677508352
0098	2.5025838929126967
0212	2.4911110473347824
0213	2.501618815026604
0216	2.0382796600257325
0218	2.4789495458562922
0224	2.5064794714133463
0231	2.4989827675638243
0233	2.498871601231687
0234	2.0361697974679025
0244	2.510779611078376
0249	2.043944390187857
0250	2.0360839397830977
0251	2.475253803572047
0252	2.0473689884871553
0254	2.4852443077628297
0255	2.51013273630489
0256	2.4988984188748
0260	2.4989502583299217
0264	2.4859335343882187
0350	2.4926186259838166
0351	2.0337690794022216
0353	2.4979496595375554
0354	2.510864019439928
0355	2.499206172839507
0357	2.4800030683151366

0358	2.4872134680610496
0359	2.5038219030949187
0370	2.4918560404714354
0371	2.4867651900770116
0373	2.492572458091805
0380	2.5071121600868955
0381	2.036961358313819
0385	2.4932353350013887
0421	2.0362377673770946
0591	2.501568916416064
0593	2.487735290310051
0598	2.4952199832233983
0685	2.0374625031173155
0856	2.4922963061488135
0886	2.031443573017604
0961	2.49063550300357
0962	2.4963914438921306
0963	2.034627086984145
0964	2.4859001098752875
0965	2.478903609491847
0967	2.505027757934287
0974	2.031594343546376
0976	2.4933139269406377
0977	2.490360560093357
0995	2.5045553308830044

3.3 Reactive programming

Not attempted

3.4 Discover Docker Containers

I used a docker in the backend of my password helper APK. Please refer to 3.5 for details and design sketch. For additional details, also see section 3.3 from my P6 report.

3.5 Enhance Password Helper (3 hours)

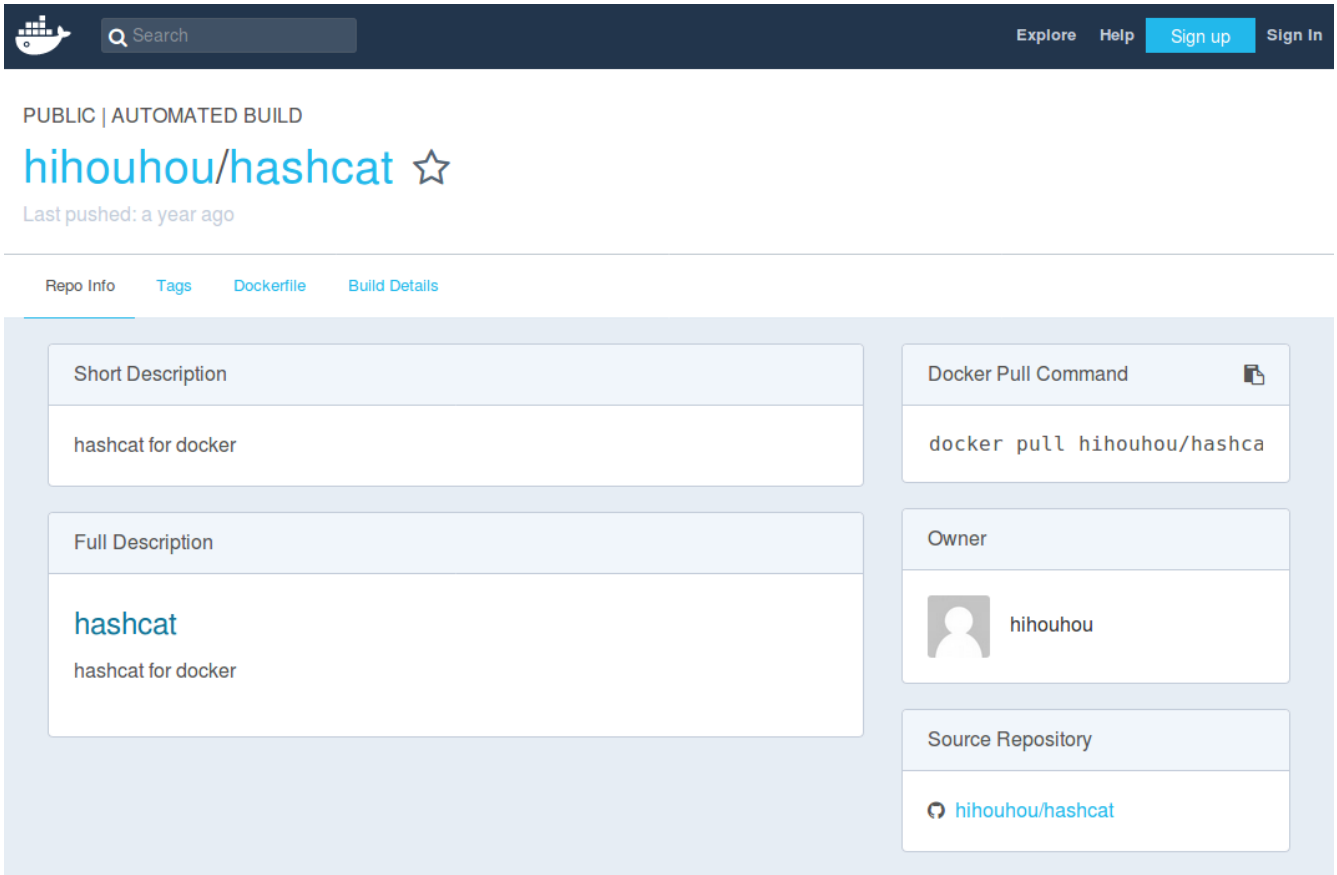
This will be created using an expansion of my P6 password APK. I will add a new activity to the password security APK which allows the user to try and crack the MD5 of his password using hashcat and word lists/rules determined by the server. I plan to use a setup similar to my hashcat docker from P6 as the password cracking tool.

To suggest pass phrases, I will include relevant links in my learn about passwords screen.

Hashcat Docker Server

In order to utilize hashcat in the cloud, I found a docker image of linux with hashcat installed and used it as my backend. This way, I do not have to worry about installing opencl or drivers dorectly on my AWS virtual machine since this proved to by very hard to do in P6.

The docker container I used was hihouhou/hashcat



This setup worked by running a server from inside the docker container and forewarding port 8080 from my AWS instance to the container using the command ‘docker run -it -p8080:8080 hihouhou/hashcat’

Next, I copied my Java server to the running container using docker cp. I installed Java on the container and ran the server.

Java Server on Hashcat

My java server is based off the one from P6 but with a few minor changes. It no longer takes two urls as input, but the plaintext password from my password helper app. It then computes the MD5 hash of this plaintext and daves it to a file. The wordlist used is rockyou-75.txt from github/SecLists. The word list id downloaded automatically by the java server if it is not already on the server. The smippet below shows the code for downloading the word list.

```
                if(new File("wordlist.txt").exists() == false)
                {
                        String wordlisturl = "https://raw.githubusercontent.com/danielmie
ssler/SecLists/master/Passwords/rockyou-75.txt";
                        //download word list file
                        URL wordlistfileurl = new URL(wordlisturl);
                        String wordlist = new Scanner(wordlistfileurl.openStream(),
                                "UTF-8").useDelimiter("\\A").next();
                        Files.write(Paths.get("./wordlist.txt"), wordlist.getBytes());

                }
```

Below is the code to generate the md5 of the user's plaintext:

```
                        //hash string and write cyphertext to a file
                        Files.write(Paths.get("./hashes.txt"), md5(plaintext).getBytes())
;
```

The md5 method is copied from https://dzone.com/articles/android-snippet-making-md5.

Finally, hashcat is run using the wordlist and T0XlC.rule rule file:

```
                        //start the process
                        Process hashcat = Runtime.getRuntime().exec("hashcat -a 0 -r T0XlC.rule "+
                                "./hashes.txt ./wordlist.txt");

                        //read the output from hashcat
                        Scanner s = new Scanner(hashcat.getInputStream());
                        output = "";
                        while(s.hasNextLine())
                                output += s.nextLine() + "\n";
```

Since the hashcat –-show option was not working properly (it would output nothing but show the password was cracked when run without -–show), I had to display the whole output from hashcat. The user can check whether the password is cracked by checking the Recovered field of hashcat's output. If it displays 1/1 (100.00%), it means the password was cracked.

```
Status...........: Cracked
Hash.Type........: MD5
Hash.Target......: 7c6a180b36896a0a8c02787eeafb0e4c
Time.Started.....: Sun Apr 23 22:45:52 2017 (0 secs)
Time.Estimated...: Sun Apr 23 22:45:52 2017 (0 secs)
Guess.Base.......: File (./wordlist.txt)
Guess.Mod........: Rules (T0XlC.rule)
Guess.Queue......: 1/1 (100.00%)
Speed.Dev.#1.....:   597.9 MH/s (6.27ms)
Recovered........: 1/1 (100.00%) Digests, 1/1 (100.00%) Salts
```

Android APK Modifications

I created a new activity CrackActivity which sends a plaintext password to the AWS/docker server and displays the server's output on screen.

The activity is almost a clone of the word list activity and does the following things:

1. onCreate registers a broadcast receiver to listen for callbacks from the QueryPassSecurity IntentService.

```
receiver =
         new BroadcastReceiver() {
             @Override
             public void onReceive(Context context, Intent intent) {
                 outputView.setText(intent.getStringExtra("output"));
             }
         };
getApplicationContext().registerReceiver(receiver,
         new IntentFilter(QueryPassSecurityService.PASSWORD_CRACK_FINISHED));
```

2. When the button is pressed, an intent containing the password is sent to QueryPassSecurityService

```
Intent serviceIntent = new Intent(this, QueryPassSecurityService.class);
serviceIntent.putExtra("type", QueryPassSecurityService.TYPE_CRACK);
serviceIntent.putExtra("password", password.getText().toString());
startService(serviceIntent);
```

3. When QueryPassSecurityService calls back to the broadcast receiver, its output is displayed on screen

See receiver code above. Sets output view to the extra "output" from the service

j4. onDestroy unregisters this broadcast receiver

```
protected void onDestroy() {
    getApplicationContext().unregisterReceiver(receiver);
    super.onDestroy();
}
```

I expanded QueryPassSecurityService to query the servers for both the word list and password cracker activiies. This was done by adding two TYPE constants which are passed as extras by the calling Activity. There are also two different FINISHED Strings which are used as the Action of the callback. This allows the WordListActivity and CrackActivity to use different IntentFilters so they only receive callbacks for the commands they sent to the service.

Constants in QueryPassSecurityService

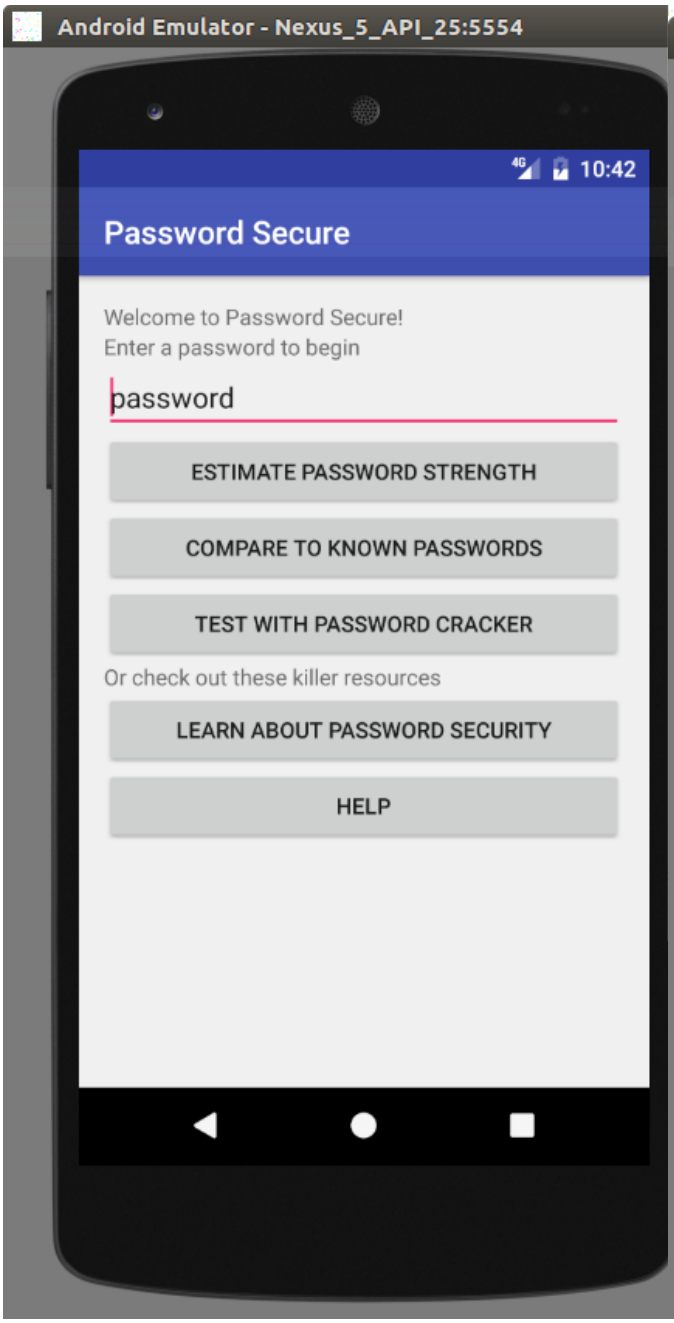
```
public static final String PASSWORD_CHECK_FINISHED =
"com.example.frodo.PASSWORD_CHECK_FINISHED";
public static final String PASSWORD_CRACK_FINISHED =
"com.example.frodo.PASSWORD_CRACK_FINISHED";
public static final String TYPE_WORD_LIST_CHECK = "check";
public static final String TYPE_CRACK = "crack";
```

Callback sets intent action based on type extra

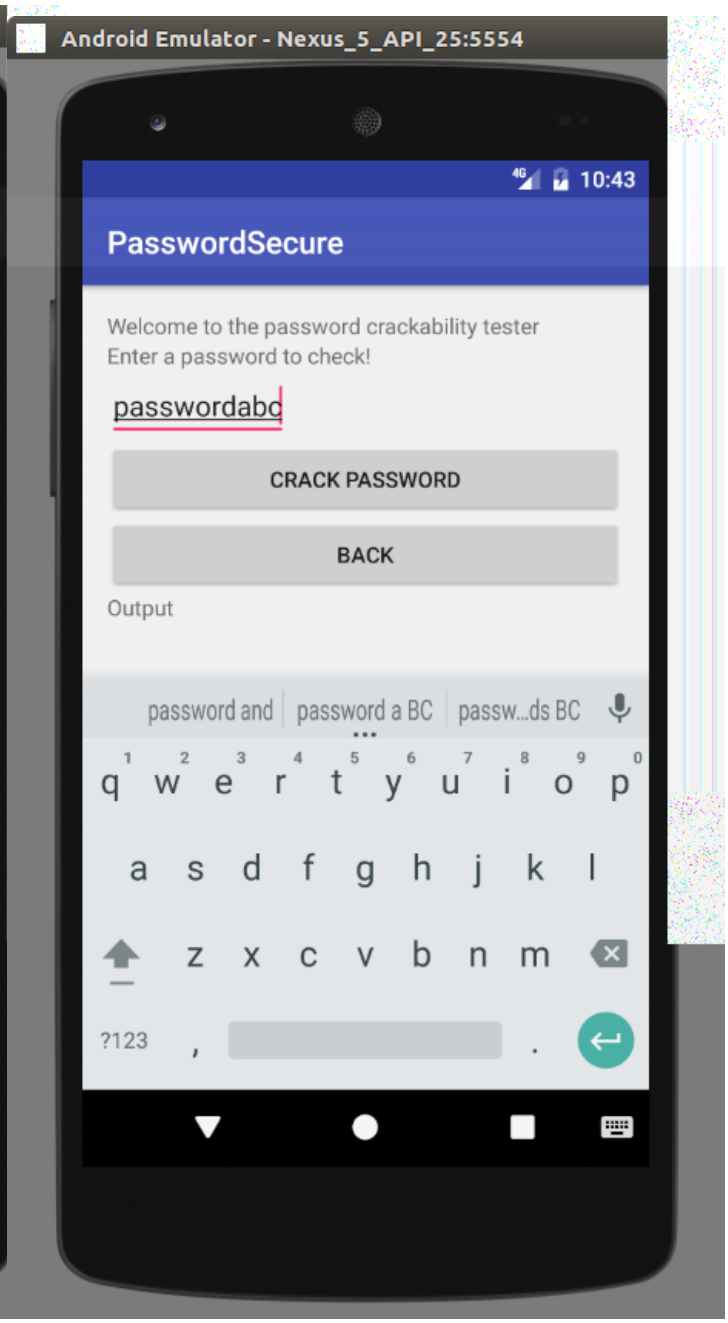
```
String type = intent.getStringExtra("type");

Intent notify = new Intent();
notify.putExtra("output",output);
if(type.equals(TYPE_WORD_LIST_CHECK))
    notify.setAction(PASSWORD_CHECK_FINISHED);
else notify.setAction(PASSWORD_CRACK_FINISHED);
sendBroadcast(notify);
```

Screenshots

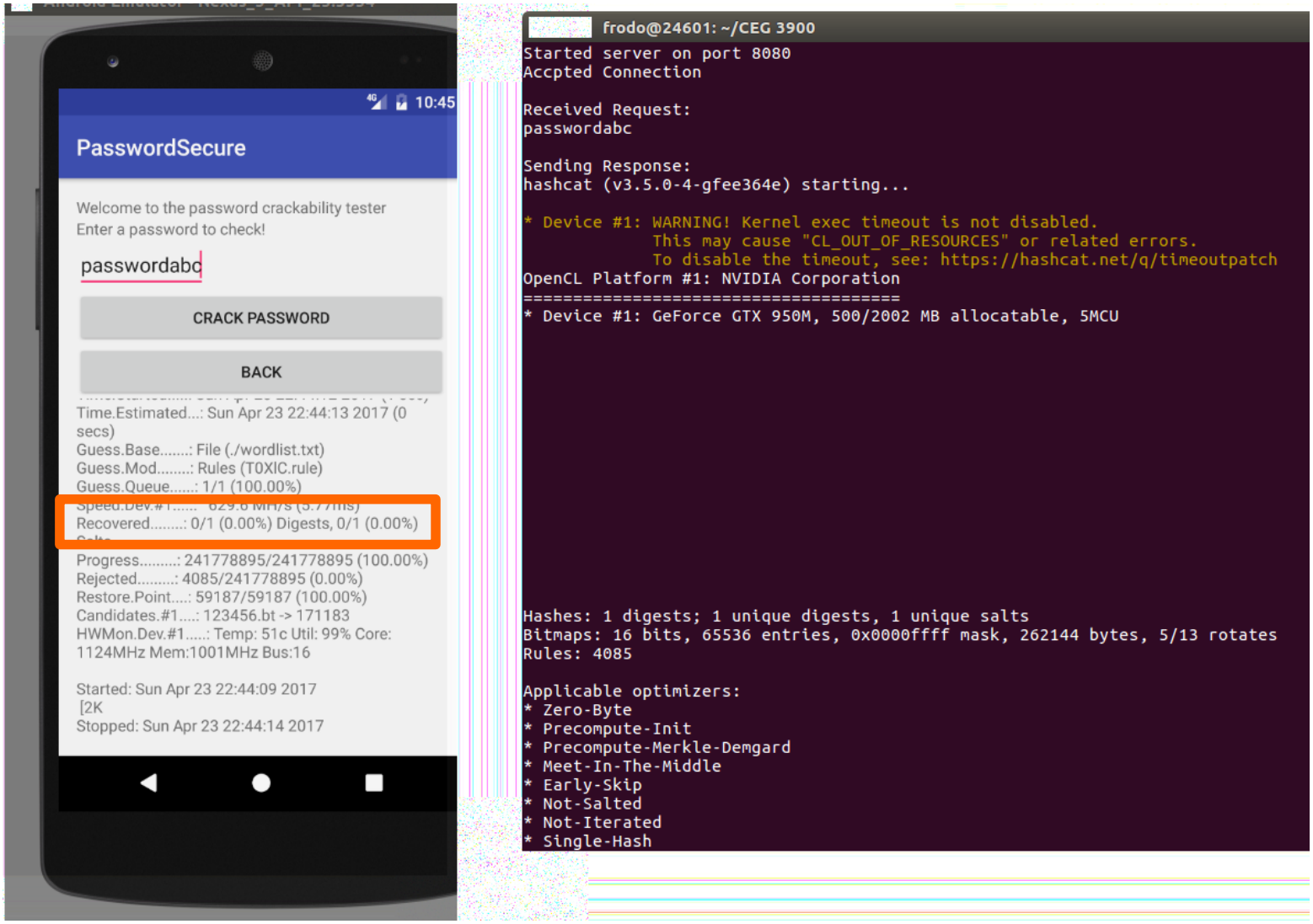
New initial screen with password crack button



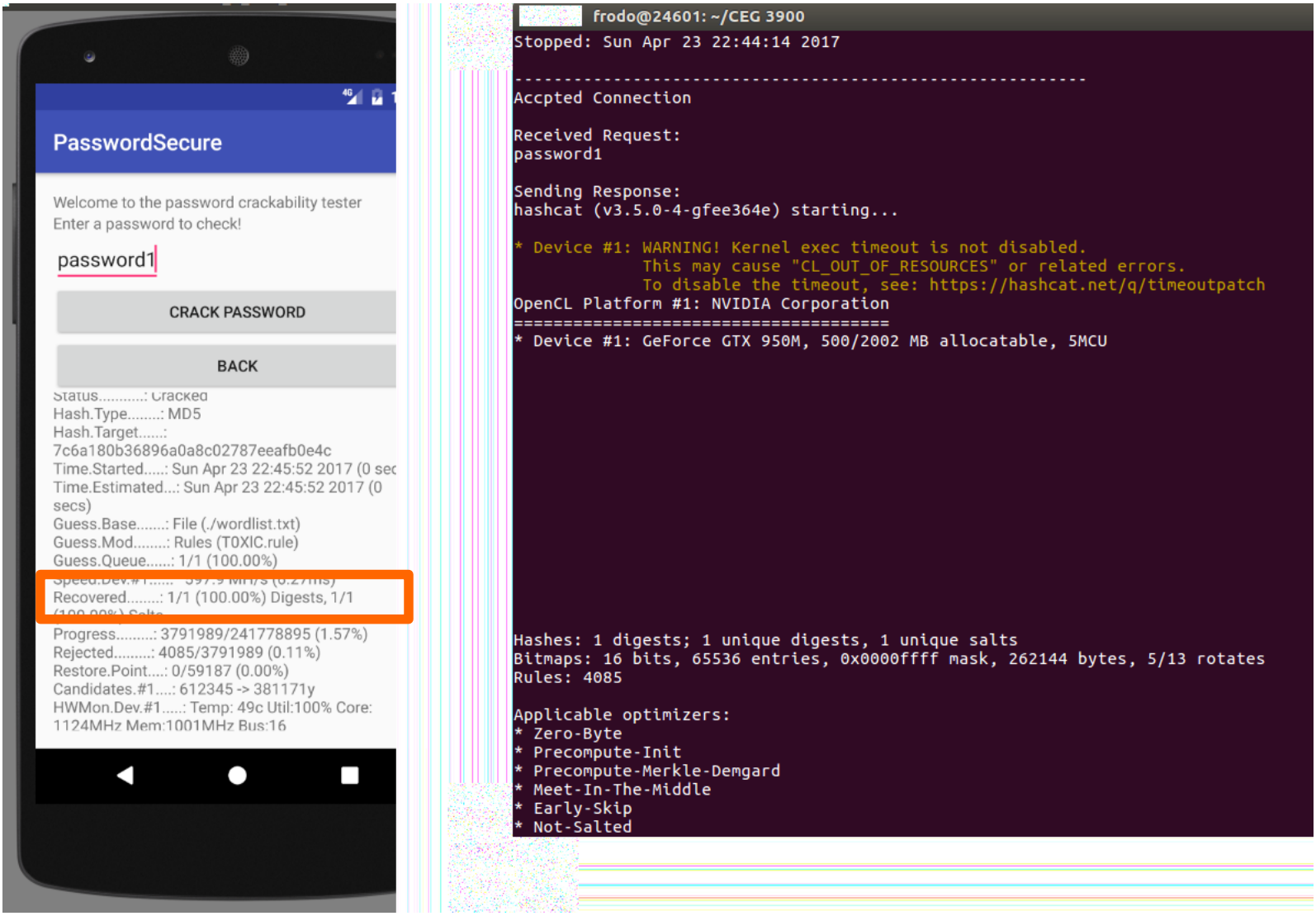
Screenshot of the password cracker activity



This screenshot shows a password that was not cracked. My cracking setup does not seem very good as passwordabc should be quite easy to crack but was not.

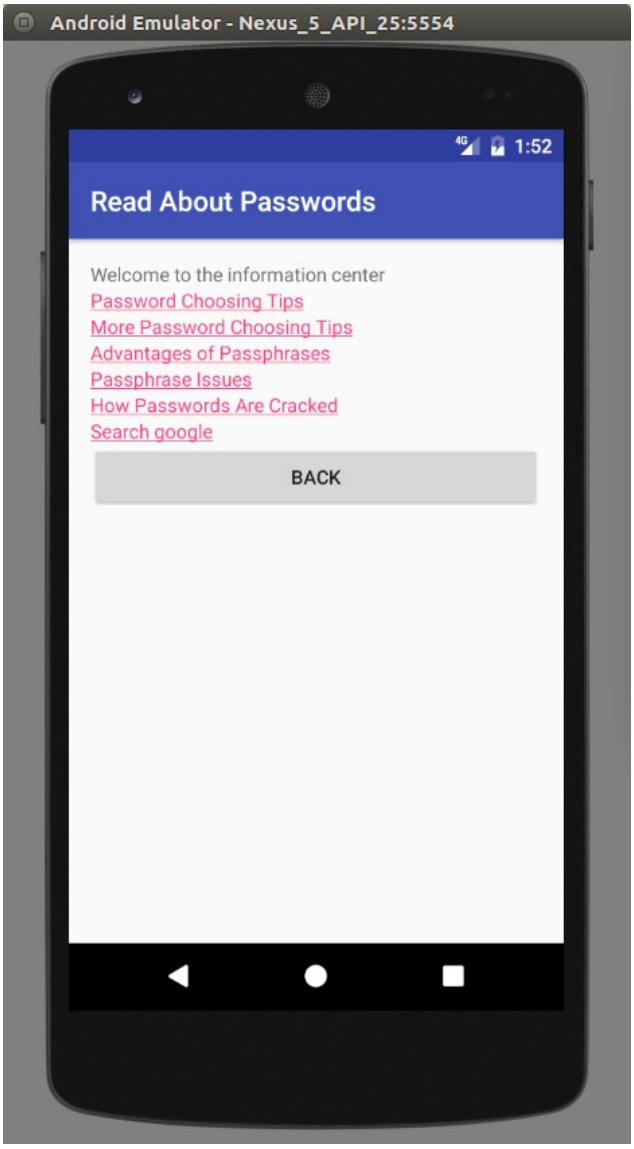


Screenshot showing a password that was cracked

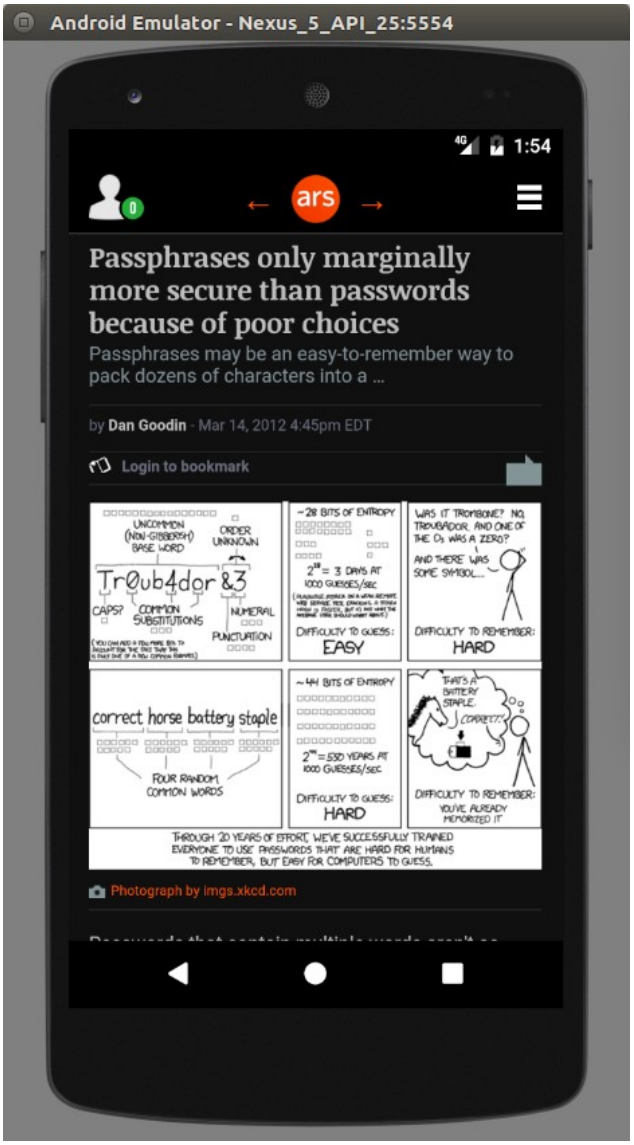


Learn about passwords activity with additional passphrase links added. This activity is accessed by pressing ‘Learn About Password Security’ on the main screen of the app.

Password security related links



Passphrase Issues link article



Complete list of links included in the APK:

https://www.schneier.com/blog/archives/2014/03/choosing_secure_1.html
https://www.cs.cmu.edu/~help/security/choosing_passwords.html
https://arstechnica.com/security/2013/05/how-crackers-make-minced-meat-out-of-your-passwords/
http://google.com/search?q=password+security
https://theintercept.com/2015/03/26/passphrases-can-memorize-attackers-cant-guess/
https://arstechnica.com/business/2012/03/passphrases-only-marginally-more-secure-than-passwords-because-of-poor-choices/